CSCI 576 Assignment 2

Instructor : Parag Havaldar
Assigned on 02/17/2023, Solutions due Friday 03/10/2023 before noon midday

Theory Part for Practice (No need to submit, solutions provided after deadline)

Question1: Color Theory

A Rubik's cube is a cube-shaped puzzle with 6 different 3x3 colored tiled sides: white, green, red, blue, orange, and yellow. The goal of the puzzle is to rotate sides and make each face have 3x3 tiles of the same color. When held under different colored lights (white, red, green, blue) the cube looks very interesting and vivid, see below:

- Explain why this happened. Why do some tiles look bright, almost glowing, while others appear muted and devoid of their original color?
- Assuming ideal conditions, you have the following lighting conditions to solve the puzzle – under pure yellow light or under red light. Which of these two light choices make it harder to solve? Give reasons for your choice of answer.

Question 2: Color Theory

- The chromaticity diagram in (x, y) represents the normalized color matching functions X, Y and Z. Prove that

  $$Z = [\ (1-x-y)/y\ ]\ Y$$

  Here you are tasked with mapping the gamut of a printer to that of a color CRT monitor. Assume that gamuts are not the same, that is, there are colors in the printer's gamut that do not appear in the monitor's gamut and vice versa. So, in order to print a color seen on the monitor you choose the nearest color in the gamut of the printer. Answer the following questions
  - Comment (giving reasons) whether this algorithm will work effectively?
  - You have two images – a cartoon image with constant color tones and a real image with varying color tones? Which image will this algorithm perform better – give reasons?
  - Can you suggest improvements rather than just choosing the nearest color?

Question 3: Entropy Coding

Consider a communication system that gives out only two symbols X and Y. Assume that the parameterization followed by the probabilities are $P(X) = x^k$ and $P(Y) = (1-x^k)$

- Write down the entropy function and plot it as a function of x for k=2. (*1 points)*
- From your plot, for what value of x with k=2 does H become a minimum? (*1 points)*
- Your plot visually gives you the minimum value of x for k=2, find out a generalized formula for x in terms of k for which H is a minimum *(3 points)*.
- From your plot, for what value of x with k=2 does H become a maximum? (*1 points)*
- Your plot visually gives you the maximum value of x for k=2, find out a generalized formula for x in terms of k for which H is a maximum *(4 points)*.

Question 4: Huffman Coding/Entropy

Bob has a pen pal, Alice, who has been learning about information theory and compression techniques. Alice decides from now on that they should exchange letters as encoded signals so they can save on ink. The following is a letter that Alice sends to Bob on her trip to Paris:

> Dear Bob,
> *Hello from Paris!*
> *I got this postcard from the Louvre. You would love Paris! I hope to hear from you.*
> Alice

- Find and show a Huffman code for the body of Alice's postcard (i.e. exclude "Dear Bob" and "Alice"). Treat each word as a symbol, and don't include punctuation. What is the average code length? (*3 points*)

Bob, having just learned about the telegram in history class, suggests to Alice that they can try writing their letters as telegram messages to shorten them even more. He sends Alice what her postcard might look like as a telegram:

> IN PARIS POSTCARD FROM LOUVRE *STOP*
> YOU WOULD LOVE *STOP*
> HOPE HEAR FROM YOU *STOP*

- Find a Huffman code for the telegram message. What is the average code length? How does it compare to the original letter? (*3 points)*
- Which version of the message, postcard, or telegram, contains more information? Show quantitatively and explain qualitatively where the difference (if any) comes from. (*4 points)*

Programming Part for submission
Topic: Compression using Vector Quantization (100 points)

This assignment will increase your understanding of image compression. We have studied JPEG compression in class, which works by transforming the image to the frequency domain and quantizing the frequency coefficients in that domain. Here you will implement a common but contrasting method using "vector quantization". Quantization or more formally scalar quantization, as you know, is a way to represent (or code) one sample of a continuous signal with a discrete value. Vector quantization on the contrary codes a group or block of samples (or a vector of samples) using a single discrete value or index.

Why does this work, or why should this work? Most natural images are not a random collection of pixels but have very smooth varying areas – where pixels are not changing rapidly. Consequently, we could pre-decide a codebook of vectors, each vector represented by a block of two pixels (or four pixels etc) and then replace all similar looking blocks in the image with one of the code vectors. The number of vectors, or the length of the code book used, will depend on how much error you are willing to tolerate in your compression. More vectors will result in larger coding indexes (and hence less compression) but results are perceptually better and vice versa. Thus vector quantization may be described as a lossy compression technique where groups or blocks of samples are given one index that represents a code word. In general this can work in k dimensions, but we will limit your implementation to two dimensions and perform vector quantization on an image.

When forming vector quantization, you need to decide on the size or type of vector you will use and the also number of vectors (which forms your codebook). For your assignment you will take as two input a parameter *M* and *N*, where the former is the size of the vectors, and the latter is the number of vectors in the code book. For the purpose of this assignment let *M*=2 which suggests that is your vectors are *two adjacent pixels* side by side. You may assume this is a *N* (number of vectors) is a power of 2 and thus after quantization each vector will need an index with log*N* bits. Your code will be called as follows and will result in a side by side display of the original image and your result after vector compression and decompression.

MyCompression.exe myImage.rgb 2 N

Here are the steps that you need to implement to compress an image with *M=2*.

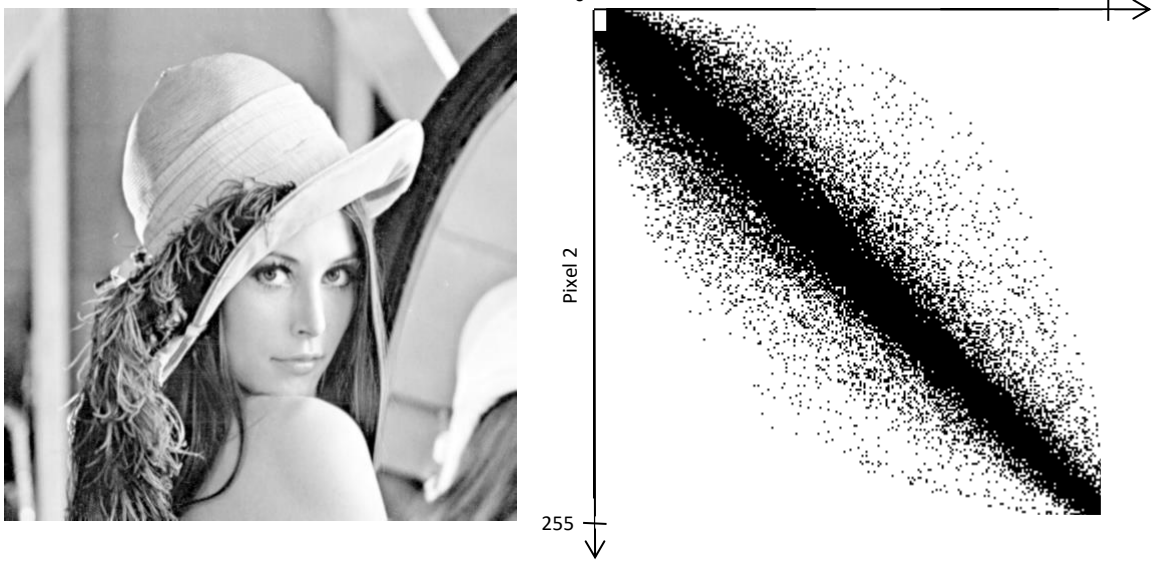
1. Understanding your two-pixel vector space to see what vectors your image contains
2. Initialization of codewords - select N initial codewords
3. Clustering vectors around each code word
4. Refine and Update your code words depending on outcome of 3.

Repeat steps 3 and 4 until code words don't change or the change is very minimal.

5. Quantize input vectors to produce output image

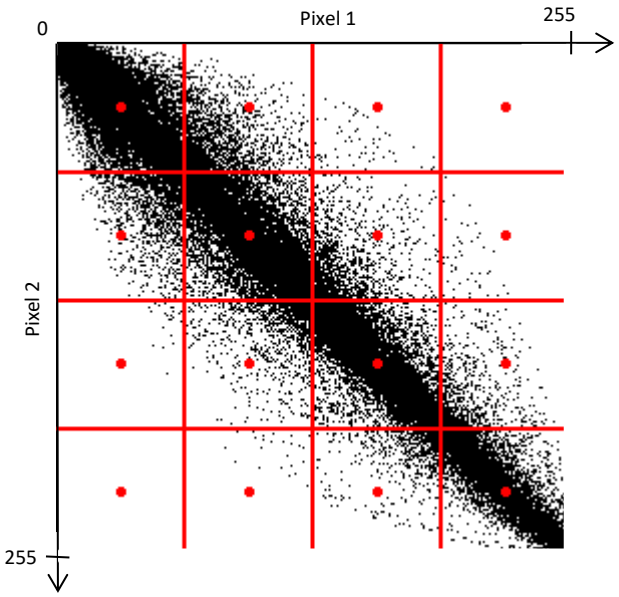
Step 1 - Understanding your vector space:

Let's look at the traditional Lena image below on the left. When creating a code book we need to decide two things – the size of the vector and the number of vectors. We have already established that we want to create a codebook of 2 pixel vectors. The space of all possible 2 pixel vectors is 256x256, and obviously not all of them are used. The right image shows a plot of which 2 pixels vectors are present in your image. Here a black dot shows a vector [pixel1, pixel2] being used in the image. This naturally is a sparse set because all combinations of [pixel1, pixel2] are not used.



Step 2 - Initialization of codewords

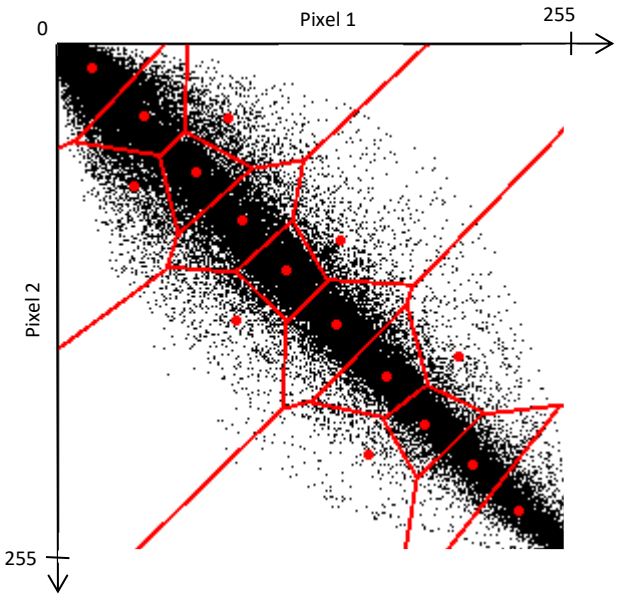
Next we need to choose the *N* vectors of two pixels each that will best approximate this data set, noting that a possible best vector may not necessarily be present in the image but is part of the space. We could initialize codewords using a heuristic, which may help cut down the number of iterations of the next two steps or we may choose our initial codewords in a random manner. The figure below shows a uniform initialization for N=16 where the space of vectors is broken into 16 uniform cells and the center of each cell is chosen as the initial codeword.



Every pair from the input image pixels would be mapped to one of these red dots during the quantization. In other words - all vectors inside a cell would get quantized to the same codebook vector. Thus compression is achieved by mapping 2 pixels in the original image to an index which needs log(N) bits or 4 bits in this example. In other words we are compressing down to 2 bits per pixel.

While the compression ratio is good, we see why this quantization is very inefficient: Two of the cells are completely empty and four other cells are very sparsely populated. The codebook vectors in the six cells adjacent to the x = y diagonal are shifted away from the density maxima in their cells, which means that the average quantization error in these cells will be unnecessarily high. Thus, six of the 16 possible pairs of pixel values are wasted, six more are not used efficiently and only four seem probably well used. This results in large overall quantization error for all codewords, also known as the mean quantization error. The next steps aim to reduce this overall mean quantization error.

Step 3 and 4:
The figure below show a much better partitioning and assignment of codewords, which is how vector quantization should perform. The cells are smaller (that is, the quantization introduces smaller errors) where it matters the most—in the areas of the vector space where the input vectors are dense. No codebook vectors are wasted on unpopulated regions, and inside each cell the codebook vector is optimally spaced with regard to the local input vector density.



This is done iteratively performing steps 3 and 4 together. Normally, no matter what your starting state is, uniformly appointed codewords or randomly selected codewords, this step should converge to the same result.

In step 3, you want to clusterize all the vectors, ie assign each vector to a codeword using the Euclidean distance measure. This is done by taking each input vector and finding the Euclidean distance between it and each codeword. The input vector belongs to the cluster of the codeword that yields the minimum distance.

In step 4 you compute a new set of codewords. This is done by obtaining the average of each cluster. Add the component of each vector and divide by the number of vectors in the cluster. The equation below gives you the updated position of each codeword y_i as the average of all vectors x_{ij} assigned to cluster i, where m is the number of vectors in cluster i.

$$y_i = \frac{1}{m} \sum_{j=1}^{m} x_{ij}$$

Steps 3 and 4 should be repeated, updating the locations of codewords and their clusters iteratively until the codewords don't change or the change in the codewords is small.

Step 5 – Quantize input vectors to produce output image:
Now that you have your code vectors, you need to maps all input vectors to one of the codewords and produce your output image. Be sure to show them side by side.

Extra Credit:
Extend the implementation for M = perfect square (30 points)

While the vector size (and hence shape) can be something of your choice, we limited this choice to *M*=2 (two side by side pixels) for the main implementation. Here you are asked to extend the implementation to accommodate perfect square for vectors eg – *M* = 4 (2x2 blocks), 9 (3x3 blocks) etc. Given a value for M, and N, your output should pick out the best and most appropriate *N* square block, the size of the square will depend on *M*. You will need to follow steps 1 through 5, but advance your algorithm to take care of higher dimensional representations.